Dokumentacja – Budzik

Grupa E04, środa 12:00

Sprzęt: LPC1769

Zakres projektu:

|  |  |  |
| --- | --- | --- |
| Funkcjonalności | Stan | Osoba odpowiedzialna |
| Pzerwania - Timer | Działający | Bartosz Żuber |
| SPI | Działający | Bartosz Żuber |
| DAC | Działający | Bartosz Żuber |
| RTC | Działający | Jakub Łysiuk |
| OLED | Działający | Jakub Łysiuk |
| Diody LED – I2C (pca9532) | Działający | Bartłomiej Rutowicz |
| Interfejs I2C | Działający | Bartłomiej Rutowicz |
| GPIO - Joystick | Działający | Bartłomiej Rutowicz |

Wykorzystane funkcjonalności

MCU: SPI, Timer, I2C

Urządzenia peryferyjne: wyświetlacz OLED, RTC, GPIO, DAC, diody LED.

1. **Instrukcja użytkownika**

Urządzenie po podłączeniu do zasilania wyświetla na ekarnie OLED aktualną datę i godzinę oraz godzinę alarmu (domyślnie ustawioną na 0:00).

Alarm jest ustawiany za pomocą joysticka. Wyborem nad zmianą godziny bądź minuty alarmu decydujemy poprzez ruch joystickiem w lewo (wybór godziny) bądź w prawo (wybór minuty), co pozwala na ruch w górę i w dół w celu zmiany wartości dla wybranej jednostki czasu. Aby zatwierdzić wybrana godzinę alarmu należy wcisnąć joystick.

Jeśli ustawiona godzina alarmu potwierdzona wciśnięciem joysticka będzie równa godzinie na zegarze urządzenie zacznie odgrywać melodie oraz włączać zestaw zielonych i czerwonych LED-ów. W celu zatrzymania alarmu należy wcisnąć joystick na dłuższą chwile (ok. 1s) a następnie go puścić.

**2. Opis algorytmu:**

1. Inicjalizacja modułów i konfiguracja pinów, takich jak SSP (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), ADC (Analog-to-Digital Converter), UART (Universal Asynchronous Receiver-Transmitter), DAC (Digital-to-Analog Converter), oraz RTC (Real-Time Clock).
2. Inicjalizacja i konfiguracja wyświetlacza OLED.
3. Inicjalizacja i konfiguracja joysticka oraz PCA9532 (8-bitowy kontroler wejścia/wyjścia).
4. Ustawienie czasu RTC na podstawie aktualnego czasu systemowego.
5. Wyświetlenie daty, godziny i informacji o alarmie na wyświetlaczu OLED.
6. Inicjalizacja i konfiguracja DAC w celu odtwarzania dźwięku.
7. Odczytanie pliku dźwiękowego z tablicy "sound\_8k" i ustalenie parametrów próbkowania.
8. Pętla główna programu:
   * Odczytanie aktualnego czasu z RTC.
   * Wyświetlenie wartości odczytanej z BNC na wyświetlaczu OLED.
   * Sprawdzenie, czy czas na RTC się zmienił, jeśli tak, zaktualizowanie odpowiednich zmiennych.
   * Wykonanie operacji obsługi joysticka, w tym odczytanie położenia joysticka, obsługa alarmu i zmiana kierunku.
   * Sterowanie diodami LED na podstawie odczytów z joysticka.
   * Odtwarzanie dźwięku na podstawie danych odczytanych z pliku dźwiękowego.
   * Powtarzanie powyższych operacji z określonym opóźnieniem czasowym.
9. Powrót do punktu 4 i kontynuacja działania programu w pętli głównej.
10. **Opis funkcjonalności**

3.1 GPIO (Joystick)

GPIO (General Purpose Input/Output) to ogólny interfejs wejścia/wyjścia. Pozwala na komunikacje i kontrole zewnętrznych urządzeń i modułów poprzez manipulacje stanem logicznym na poszczególnych pinach. GPIO to uniwersalny interfejs który może zostać skonfigurowany jako wejście lub wyjście w zależności od potrzeb. Każdy pin może mieć własne ustawienia konfiguracji, takie jak tryb pracy czy przerwania. W naszym przypadku joystick służy do wyboru godziny alarmu, zatwierdzenia jej i wyłączenia alarmu (dokładne działanie w instrukcji użytkownika).

**Rejestry GPIO:**

- **FIODIR** - Rejestr kierunku. Pozwala na ustawienie pinu jako wiejście lub wyjście ( bit 1 oznacza wejście 0 oznacza wyjście)

- **FIOPIN** – Rejestr danych. Umożliwia odczytanie lub zapisanie wartości logicznej na pinie. Bit ustawiony na 1 oznacza stan wysoki, a na 0 oznacza stan niski.

- **FIOSET** – Rejestr służący do ustawiania pinów na stan wysoki (Wpisanie 1 w rejestrze ustawi pin na wysoki, 0 nie ma efektu).

- **FIOCLR** – Rejestr służący do ustawiania pinów na stan niski (Wpisanie 1 w rejestrze ustawi pin na niski, 0 nie ma efektu). .

Inicjalizacja komunikacji za pomocą joysticka dokonywana jest poprzez funkcje void joystick\_init(void) w której za pomocą funkcji PIO\_SetDir(uint8\_tportNum, uint32\_t bitValue, uint8\_t dir), wybierany jest port i piny, które będą służyły jako wejście dla joysticka przy czym do rejestru FIODIR jest wprowadzana odpowiednia wartość.

void GPIO\_SetDir(uint8\_t portNum, uint32\_t bitValue, uint8\_t dir)

{

    LPC\_GPIO\_TypeDef \*pGPIO = GPIO\_GetPointer(portNum);

    if (pGPIO != NULL) {

        // Enable Output

        if (dir) {

            pGPIO->FIODIR |= bitValue;

        }

        // Enable Input

        else {

            pGPIO->FIODIR &= ~bitValue;

        }

    }

}

Piny wykorzystywane do kontroli joysticka: P0,15| P0,16 | P0,17 | P2,3 | P2,4 |

void joystick\_init (void)

{

    /\* set the GPIOs as inputs \*/

/\*

    GPIO\_SetDir( 0, 15, 0 );

    GPIO\_SetDir( 0, 16, 0 );

    GPIO\_SetDir( 0, 17, 0 );

    GPIO\_SetDir( 2, 3, 0 );

    GPIO\_SetDir( 2, 4, 0 );

\*/

    GPIO\_SetDir( 0, (1<<15|1<<16|1<<17), 0 );

    GPIO\_SetDir( 2, (1<<3|1<<4), 0 );

}

Odczytanie stanu portu jest wykonywane za pomocą funkcji uint8\_t joystick\_read(void), w której zawarta jest funkcja GPIO\_ReadValue(uint8\_tportNum) odczytująca stan portu z rejestru FIOPIN. Jeśli joystick zostanie przesunięty w funkcja joystick\_read zwróci nam jego aktualny stan.

3.2 Interfejs I2C

I2C (Inter-Integrated Circuit) jest interfejsem komunikacyjnym używanym do komunikacji między różnymi układami elektronicznymi w systemach wbudowanych. Składa się z dwóch linii SDA (Serial Data Line) i SCL(Serial Clock Line). Linia SDA służy do przesyłania danych między urządzeniami, a linia SCL służy do synchronizacji komunikacji poprzez generowanie impulsów zegarowych. W protokole I2C wyróżnia się urządzenia typu master i slave. Urządzenie master inicjuje komunikacje i przesył danych, urządzenie slave reaguje na polecenia i przekazują informacje zgodnie z rządaniami urządzenia master.

**Rejestry I2C:**

**- I2CONSET –** Rejestr kontrolny do ustawiania bitów sterujących w module I2C. Może zostać użyty do ustawienia flag: **STA** (Ustawienie tej flagi spowoduje wygenerowanie sygnału START i rozpoczęcie transmisji) , **STO** (Ustawienie tego bitu powoduje wygenerowanie sygnału STOP i zakończenie transmisji), **SI** (bit używany do obsługi przerwań), **AA** (Ustawienie tego bitu powoduje wysłanie potwierdzenia ACK do urządzenia slave. Ustawienie 0 powoduje wysłanie NACK) i **I2EN** (flaga służąca do włączenia modułu I2C.) .

**- I2STAT –** Rejestr zawierający informacje o stanie transmisji I2C. Może służyć do sprawdzania wyników operacji (stanów kodu), takich jak wysyłanie i otrzymywanie danych czy wystąpień kolizji.

**- I2CONCLR –** Służy do czyszczenia ustawień flag sterujących w module I2C. Przyjmuje flagi **STAC** (Ustawienie tego bitu spowoduje wyczyszczenie flagi STA, co oznacza zakończenie sygnału START) i **I2ENC** (Ustawienie tego bitu spowoduje wyłączenie interfejsu I2C).

**- I2SCLH –** Rejestr służący do ustawienia czasu trwania cyklu wysokiego na linii SCL.

**- I2SCLL -** Rejestr służący do ustawienia czasu trwania cyklu niskiego na linii SCL.

**- I2DAT0/1/2/3 –** Rejestr przechowujący dane.

**- I2ADR0/1/2/3 –** Rejestr przechowujący adresy slave.

**Inicjalizacja I2C**

static void init\_i2c(void)

{

PINSEL\_CFG\_Type PinCfg;

/\* Initialize I2C2 pin connect \*/

PinCfg.Funcnum = 2;

PinCfg.Pinnum = 10;

PinCfg.Portnum = 0;

PINSEL\_ConfigPin(&PinCfg);

PinCfg.Pinnum = 11;

PINSEL\_ConfigPin(&PinCfg);

// Initialize I2C peripheral

I2C\_Init(LPC\_I2C2, 100000);

/\* Enable I2C1 operation \*/

I2C\_Cmd(LPC\_I2C2, ENABLE);

}

Rejestr LPC\_I2C2 reprezentuje moduł I2C o numerze 2 na płytce. Ustawiamy prędkość transmisji na 100kHz (100000 impulsów na sekundę na linii SCL). Ustawiamy funkcje pinu na 2 ( co oznacza funkcje I2C).

Wybierany jest port o numerze 0, i konfigurowane są piny. P0[10] jako SDA i P[11] jako SCL.

Następnie włączany jest moduł I2C2 za pomocą funkcji I2C\_Cmd(LPC\_I2C2, ENABLE);.

void I2C\_Cmd(LPC\_I2C\_TypeDef\* I2Cx, FunctionalState NewState)

{

CHECK\_PARAM(PARAM\_FUNCTIONALSTATE(NewState));

CHECK\_PARAM(PARAM\_I2Cx(I2Cx));

if (NewState == ENABLE)

{

I2Cx->I2CONSET = I2C\_I2CONSET\_I2EN;

}

else

{

I2Cx->I2CONCLR = I2C\_I2CONCLR\_I2ENC;

}

}

W przypadku włączenia modułu ustawiona zostaje flaga **I2EN** na rejestrze **I2CONSET** umożliwiając wykonywanie operacji interface’u.

3.3 DAC

Inicjalizacja pinów GPIO, które są używane do sterowania układem LM4811.

GPIO\_SetDir(2, 1<<0, 1);

GPIO\_SetDir(2, 1<<1, 1);

GPIO\_SetDir(0, 1<<27, 1);

GPIO\_SetDir(0, 1<<28, 1);

GPIO\_SetDir(2, 1<<13, 1);

GPIO\_SetDir(0, 1<<26, 1);

GPIO\_ClearValue(0, 1<<27); //LM4811-clk

GPIO\_ClearValue(0, 1<<28); //LM4811-up/dn

GPIO\_ClearValue(2, 1<<13); //LM4811-shutdn

**Działanie wykorzystywanych pinów:**

* P2[0] (TXD1) - Universal Asynchronous Receiver-Transmitter. Przesyłanie danych między urządzeniami. Pin P2.0 pełni rolę linii transmitującej dane (TXD) dla interfejsu UART1.
* P2[1] (PWM1[2]) - pin wykorzystywany do modulacji szerokości sygnału (PWM - Pulse Width Modulation). Modulacja szerokości sygnału umożliwia regulację sygnału o zmiennej szerokości impulsów. Pin P2.1 jest przypisany do modułu PWM1 i pełni rolę wyjścia modulacji PWM.
* P2[13] (EINT3) - odbieranie zewnętrznych przerwań (EINT - External Interrupt). Pin P2.13 jest skonfigurowany jako wejście przerwań, które może reagować na zewnętrzne sygnały/przerwania
* P0[26] (AOUT) - wyprowadzenie danych z przetwornika DAC (Digital-to-Analog Converter). Przetwornik ten konwertuje sygnały cyfrowe na analogowe. Pin P0.26 jest używany do przekazywania analogowego sygnału wyjściowego z przetwornika DAC.
* P0[27] (SDA1) - obsługa wejścia/wyjścia danych dla komunikacji I2C (Inter-Integrated Circuit). I2C jest szeregowym protokołem komunikacyjnym, który umożliwia komunikację między różnymi układami elektronicznymi. Pin P0.27 pełni rolę linii danych (SDA) w protokole I2C.
* P0[28] (SCL1) - obsługa zegara dla komunikacji I2C. Pin P0.28 pełni rolę linii zegara (SCL) w protokole I2C.

**Konfiguracja pinu DAC**

Konfiguracja odbywa się za pomocą rejestru PINSEL, który mapuje funkcje pinów. Pin P0.26 jest skonfigurowany jako pin wyjściowy DAC poprzez ustawienie wartości 2 w polu Funcnum. Pin jest ustawiony jako standardowy pin cyfrowy (nie open-drain), a tryb pinu jest ustawiony na wartość 0.

PinCfg.Funcnum = 2;

PinCfg.OpenDrain = 0;

PinCfg.Pinmode = 0;

PinCfg.Pinnum = 26;

PinCfg.Portnum = 0;

PINSEL\_ConfigPin(&PinCfg);

## Inicjalizacja modułu DAC:

Ustawienie odpowiednich bitów w rejestrze DACR. Pozwala to na skonfigurowanie maksymalnego prądu wyjśćiowego

DAC\_Init(LPC\_DAC);

## Sprawdzanie nagłówka pliku dźwiękowego:

Realizacja poprzez porównanie znaków w tablicy ‘sound\_8k’ na podstawie indeksu ‘cntW’. Jeśli wartości i znaki nie pasują do siebie, komunikat zostaje wysłany przez interfejs UART a funkcja zwraca wartość 0 (czyli błąd).

if (sound\_8k[cntW] != 'R' && sound\_8k[cntW+1] != 'I' &&

sound\_8k[cntW+2] != 'F' && sound\_8k[cntW+3] != 'F')

{

UART\_SendString(UART\_DEV, (uint8\_t\*)"Wrong format (RIFF)\r\n");

return 0;

}

cntW += 4;

## Ustawianie wartości opóźnienia

Obliczana jest wartość opóźnienia między kolejnymi próbkami dźwięku na podstawie wzoru 1000000 / sampleRate, co daje wartość w mikrosekundach. Ta wartość opóźnienia jest używana w dalszej części programu do regulowania odtwarzania dźwięku.

sampleRate = (sound\_8k[cntW] | (sound\_8k[cntW+1] << 8) |

(sound\_8k[cntW+2] << 16) | (sound\_8k[cntW+3] << 24));

if (sampleRate != 8000) {

UART\_SendString(UART\_DEV, (uint8\_t\*)"Only 8kHz supported\r\n");

return 0;

}

delay = 1000000 / sampleRate;

## Przygotowanie do odtwarzania danych dźwiękowych

Wartość off jest ustawiana na indeks cntW, aby wskazać na miejsce w tablicy, od którego należy rozpocząć odtwarzanie danych dźwiękowych.

off = cntW;

3.4 timer

Inicjalizacja timera

uint32\_t delay;

void Timer0\_Wait(delay);

Inicjacja Timera 0 jest dokonywana przez powyższą funkcje ustawiając odpowiednie rejestry:

* rejestr kontrolny (TCR), służący do kontrolowania trybu działania Timera 0. Odpowiada również za obsługę Timer Countera (TC), który jest 32bitowym rejestrem inkrementowanym pewien cykl.
* rejestr przerwań (IR), odpowiadający za rejestrowanie zdarzeń związanych z timerem 0, np. przekroczenie MR0 i MR1.
* rejestr preskalera(PR), ustawiający wartość preskalera odpowiedzialnego za kontrolę częstotliwości odliczania timera.
* rejestr punktu odniesienia MR0 (Match Register 0), używany do prównania z wartością Timera0. Przy osiągnięciu tej samej wartości przez Timer0 co MR0 może zostać zainicjalizowane przerwanie.

W celu konfiguracji timera w trybie Timer (TIMER\_MODE).

W następnej kolejności funkcja sprawdza wartości rejestrów timera w stałych odstępach czasu do osiągnięcia wartości określonej przez ‘delay’. Następnie inicjalizowany jest PR Timera 0. Funkcja oczekuje w pętli sprawdzając czy wartość osiągniętą przez TC odpowiada oczekiwanemu czasowi. Gdy to nastąpi funkcja zakończy działanie.

void Timer0\_us\_Wait(uint32\_t time);

Działanie podobne do Funkcji ‘Timer0\_Wait(delay)’, ale stworzony do działania w mikrosekundach i korzystający dodatkowo z rejestru:

* Rejestr punktu odniesienia MR1 (Match Register 1), który jest drugim punktem odniesienia po MR0.

Związanego z koniecznością zastosowania większej dokładności związanej z działaniem na mniejszych liczbach.

3.5 Diody LED – I2C (pca9532)

Moduł PCA9532 (inaczej nazywany **Port Expanderem**) to układ scalony implementujący dwukanałowy 8-bitowy przetwornik I/O (Input/Output). Układ ten jest obsługiwany przez interfejs I2C. W naszym przypadku jest on wykorzystywany do obsługi 16-bitowego zestawu LEDów. Moduł obsługuje 16 linii wejścia/wyjścia, podzielonych na dwa niezależne 8-bitowe kanały. Każdy z 16 pinów można konfigurować jako wejście lub wyjście.

Rozpiska układu LEDów:

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, numer

Opis wygenerowany automatycznie

**Rejestry kontrolne:**

**-INPUT0/1-** Rejestry **INPUT0 i INPUT1** służą do odzwierciedlania aktualnego stanu poszczególnych pinów (**INPUT0** odpowiada za piny o indexach 0-7, a **INPUT1** za piny o indexach 8-15). Rejestr służy tylko do odczytu, próby zapisu będą odbierane ale nie będą miały żadnego efektu.

**- PSC0/1-** Rejestr określa szybkość migania kolejno dla kanałów 0 i 1 (wykorzystywane we fladze BLINK0/1).

**- PWM0/1 –** Rejestr określa wypełnienie cyklu dla sygnału BLINK0. Wyjścia są w stanie NISKIM (dioda LED włączona), gdy wartość licznika jest mniejsza od wartości w rejestrze PWM0, a w stanie WYSOKIM (dioda LED wyłączona), gdy jest większa. Jeśli rejestr PWM0 zostanie ustawiony na wartość 00h, to wyjście PWM0 będzie zawsze WYSOKIE (dioda LED wyłączona).

**- LS0/1/2/3 –** LED selector. Rejestry sterujące diodami LED kolejno dla portów 0/1/2/3. Służy do ustawiania stanów diod LED. Każdemu LEDowi **przysługują po dwa bity** (jak na rysunku powyżej), podając wartość **00** – LED w stanie wysokiej impedancji (domyślnie LED off), **01** – LED w stanie niskim (LED on), **10** – LED będzie mrugał zgodnie z parametrami określonymi przez **PWM0, 11** – LED będzie mrugał zgodnie z parametrami określonymi przez **PWM1.**

**Konfiguracja:**

Ekspander sam w sobie nie musi zostać zainicjalizowany.

void pca9532\_init (void)

{

/\* nothing to initialize \*/

}

Kod obsługujący LEDy:

if (dir) {

if (cnt == 0)

cnt = 31;

else

cnt--;

} else {

cnt++;

if (cnt >= 32)

cnt = 0;

}

if (cnt < 16)

ledOn |= (1 << cnt);

if (cnt > 15)

ledOn &= ~( 1 << (cnt - 10) );

if (cnt > 15)

ledOff |= ( 1 << (cnt - 15) );

if (cnt < 16)

ledOff &= ~(1 << cnt)

pca9532\_setLeds(ledOn, ledOff);

Po wykonaniu odpowiednich operacji bitowych na zmiennych ledOn i ledOff, tak aby LEDy zapalały się po kolei jeden po drugim wprowadzamy te zmienne do funkcji:

void pca9532\_setLeds (uint16\_t ledOnMask, uint16\_t ledOffMask)

{

/\* turn off leds \*/

ledStateShadow &= (~(ledOffMask) & 0xffff);

/\* ledOnMask has priority over ledOffMask \*/

ledStateShadow |= ledOnMask;

/\* turn off blinking \*/

blink0Shadow &= (~(ledOffMask) & 0xffff);

blink1Shadow &= (~(ledOffMask) & 0xffff);

setLeds();

}

Funkcja ta służy do ustawiania stanu diody LED. l ledStateShadow &= (~(ledOffMask) & 0xffff); to lina kodu służąca do wyłączenia diody LED, wynik wykonanej operacji bitowej jest przechowywany w zmiennej ledStateShadow, dzięki czemu diody które mają odpowiadające bity w ledOffMask ustawione na 1, zostaną ustawione na 0 a co za tym idzie wyłączone. ledStateShadow |= ledOnMask; to linia kodu służąca do włączania diody LED zgodnie z maską ledOnMask. Następnie wywoływana jest funkcja setLeds(); aktualizująca stan diod LED.

static void setLeds(void)

{

uint8\_t buf[5];

uint8\_t ls[4] = {0,0,0,0};

uint16\_t states = ledStateShadow;

/\* LEDs in On/Off state \*/

setLsStates(states, ls, LS\_MODE\_ON);

/\* set the LEDs that should blink \*/

setLsStates(blink0Shadow, ls, LS\_MODE\_BLINK0);

setLsStates(blink1Shadow, ls, LS\_MODE\_BLINK1);

buf[0] = PCA9532\_LS0 | PCA9532\_AUTO\_INC;

buf[1] = ls[0];

buf[2] = ls[1];

buf[3] = ls[2];

buf[4] = ls[3];

I2CWrite(PCA9532\_I2C\_ADDR, buf, 5);

}

buf[5] – bufor zawierający dane do wysłania poprzez I2C

ls[4] – tablica przechowująca stany diod LED (0-wyłączona, 1-włączona)

states – zmienna przechowująca wartość ledStateShadow (stan diod LED włączonych i wyłączonych)

Wywoływana jest funkcja setLsStates która ustawia stan diody LED na podstawie wartości zmiennej status, przekazywany jest tryb LS\_MODE\_ON mówiący ze chcemy ustawić diody w trybie ON/OFF.

W buforze są zapisywane wartości mówiące o adresie pierwszej diody LED i włączeniu automatycznej inkrementacji adresu przy zapisie kolejnych rejestrów i wartości z tablicy ls przechowujące stany diod LED.

Wywoływana jest funkcja I2CWrite(PCA9532\_I2C\_ADDR, buf, 5); przesyłająca dane z bufora buf przez interfejs I2C na adres modułu PCA9532 w celu aktualizacji stanu diody LED.

static int I2CWrite(uint8\_t addr, uint8\_t\* buf, uint32\_t len)

{

I2C\_M\_SETUP\_Type txsetup;

txsetup.sl\_addr7bit = addr;

txsetup.tx\_data = buf;

txsetup.tx\_length = len;

txsetup.rx\_data = NULL;

txsetup.rx\_length = 0;

txsetup.retransmissions\_max = 3;

if (I2C\_MasterTransferData(I2CDEV, &txsetup, I2C\_TRANSFER\_POLLING) == SUCCESS){

return (0);

} else {

return (-1);

}

}

Tworzony jest obiekt txtsetup który przechowuje konfiguracje przesyłanych danych.

Następnie ustawiane są kolejno pola tej struktury: addr – adres urządzenia docelowego, buf – bufor danych do przesłania, len – długość danych do przesłania i retransmissions\_max – maksymalna liczba retransmisji danych.

Wykonuje się funkcja I2C\_MasterTransferData która inicjuje transfer danych za pomocą interfejsu I2C.

3.6 SPI

SPI (Serial Peripheral Interface) jest to szeregowy interfejs synchroniczny umożliwiający komunikacje między mikrokontrolerem (master) a urządzeniami peryferyjnymi (slave). Dane są przesyłane sekwencyjnie bit po bicie w jednym cyklu zegarowym. Na płytce LPC1769, rejestr SPI jest 8-bitowy co oznacza przesył danych w 8-bitowych bajtach.

**Konfiguracja pinów:**

PinCfg.Funcnum = 2;

PinCfg.OpenDrain = 0;

PinCfg.Pinmode = 0;

PinCfg.Portnum = 0;

PinCfg.Pinnum = 7;

PINSEL\_ConfigPin(&PinCfg);

PinCfg.Pinnum = 8;

PINSEL\_ConfigPin(&PinCfg);

PinCfg.Pinnum = 9;

PINSEL\_ConfigPin(&PinCfg);

PinCfg.Funcnum = 0;

PinCfg.Portnum = 2;

PinCfg.Pinnum = 2;

PINSEL\_ConfigPin(&PinCfg);

* SSEL –P2.2
* SCK –P0.7
* MISO –P0.8
* MOSI –P0.9

**Opisy pinów:**

1. SCK (Serial Clock):
   * Wejście/Wyjście.
   * Sygnał zegara synchronizującego transfer danych przez interfejs SPI.
   * Zawsze jest generowany przez urządzenie typu master i odbierany przez urządzenie typu slave.
   * Zegar może być programowany jako aktywny wysoki (active high) lub aktywny niski (active low).
   * Interfejs SPI jest aktywny tylko podczas transferu danych. W pozostałym czasie znajduje się w stanie nieaktywnym lub trójstanowym.
2. SSEL (Slave Select):
   * Wejście.
   * Sygnał wyboru slave'a w interfejsie SPI.
   * Sygnał SSEL jest aktywny niski (active low), co oznacza, że stan niski wskazuje, który slave jest wybrany do uczestnictwa w transferze danych.
   * Każdy slave ma swój własny unikalny sygnał SSEL.
   * SSEL musi być w stanie niskim przed rozpoczęciem transakcji danych i zazwyczaj pozostaje w tym stanie przez cały czas trwania transakcji.
   * Jeśli sygnał SSEL zmieni się na stan wysoki (high) w trakcie transmisji danych, transfer jest uważany za przerwany. W takim przypadku slave powraca do stanu bezczynności, a otrzymane dane są odrzucane. Nie ma innych sygnałów wskazujących na to zdarzenie. Sygnał ten nie jest bezpośrednio sterowany przez mastera i może być sterowany przez prosty układ ogólnego przeznaczenia (general purpose I/O) pod kontrolą oprogramowania.
3. MISO (Master In Slave Out):
   * Wejście/Wyjście.
   * Sygnał unidirectionalny służący do przesyłania danych szeregowych z slave'a do mastera w interfejsie SPI.
   * Gdy urządzenie jest slave'em, dane szeregowe są przesyłane na tym pinie jako sygnał wyjściowy.
   * Gdy urządzenie jest masterem, dane szeregowe są przyjmowane na tym pinie jako sygnał wejściowy.
   * Gdy urządzenie slave nie jest wybrane, to znaczy sygnał SSEL jest w stanie wysokim (high), slave ustawi ten sygnał w stanie wysokim impedancji.
4. MOSI (Master Out Slave In):
   * Wejście/Wyjście.
   * Sygnał unidirectionalny służący do przesyłania danych szeregowych z mastera do slave'a w interfejsie SPI.
   * Gdy urządzenie jest masterem, dane szeregowe są przesyłane na tym pinie jako sygnał wyjściowy.
   * Gdy urządzenie jest slave'em, dane szeregowe są przyjmowane na tym pinie jako sygnał wejściowy.

**Dla mikrokontrolera LPC1769, interfejs SPI korzysta z następujących rejestrów:**

1. SPI Control Register (SPCR):Rejestr kontrolny, który zawiera bity konfigurujące interfejs SPI.
   * Bit 2 (BitEnable): Ustawienie tego bitu powoduje wysłanie i odbiór 8 bitów danych podczas jednej transmisji.
   * Bit 3 (CPHA): Kontroluje fazę zegara dla przechwytywania danych. Ustawienie 0 oznacza, że dane są przechwytywane przy zboczu opadającym zegara, a ustawienie 1 oznacza, że dane są przechwytywane przy zboczu narastającym zegara.
   * Bit 4 (CPOL): Kontroluje poziom aktywny zegara w stanie bezczynności. Ustawienie 0 oznacza, że zegar jest aktywny na wysokim poziomie, a ustawienie 1 oznacza, że zegar jest aktywny na niskim poziomie.
   * Bit 5 (MSTR): Wybór trybu pracy interfejsu SPI: tryb Slave (00) lub tryb Master (01).
   * Bit 6 (LSBF): Kontroluje kolejność bitów podczas transmisji: najbardziej znaczący bit pierwszy (00) lub najmniej znaczący bit pierwszy (01).
   * Bit 7 (SPIE): Ustawienie tego bitu umożliwia generowanie przerwań przez interfejs SPI.
2. SPI Status Register (SPSR):Rejestr statusu, który zawiera bity informujące o stanie interfejsu SPI.
   * Bit 3 (ABRT): Flaga przerwanego transferu. Jest ustawiana, gdy transfer danych został przerwany z powodu wzrostu sygnału Slave Select (SSEL) w trakcie transmisji.
   * Bit 4 (MODF): Flaga błędu trybu SPI. Jest ustawiana, gdy wystąpił błąd trybu SPI, na przykład gdy Master nie otrzymał oczekiwanej odpowiedzi od Slave.
   * Bit 5 (ROVR): Flaga nadmiarowego bufora odbiornika. Jest ustawiana, gdy wystąpiło nadmiarowe zapełnienie bufora odbiornika, co oznacza utratę danych.
   * Bit 6 (WCOL): Flaga kolizji zapisu. Jest ustawiana, gdy wystąpiła kolizja zapisu do rejestru danych SPI.
   * Bit 7 (SPIF): Flaga poziomu SPIF. Jest ustawiana, gdy flaga SPIF osiągnęła ustalony poziom.
3. SPI Data Register (SPDR):Rejestr danych, który służy do zapisywania i odczytywania danych przesyłanych przez interfejs SPI. W przypadku transmisji z mikrokontrolera do urządzenia zewnętrznego, dane są zapisywane do tego rejestru przed rozpoczęciem transmisji. W przypadku odczytu danych z urządzenia zewnętrznego, zapisane dane są odczytywane z tego rejestru po zakończeniu transmisji.
4. SPI Clock Counter Register (SPCCR):Rejestr licznika zegarowego, który pozwala na ustawienie częstotliwości zegara SPI. Zawiera flagę przerwania (bit flagi w rejestrze FLAGS procesora, który określa, czy procesor będzie natychmiast reagował na maskowalne przerwania sprzętowe.) dla interfejsu SPI. Ustawienie wartości tego rejestru pozwala na kontrolę prędkości transmisji SPI.
5. SPI Interrupt Register (SPINT):rejestr kontroluje przerwania generowane przez interfejs SPI i zawiera bity ustawiane w celu zgłoszenia przerwania lub odczytu statusu przerwania.
   * Bit 0 (SPIF): Bit przerwania transferu SPI. Ustawiany, gdy transfer danych przez interfejs SPI jest zakończony. Przerwanie to jest zgłaszane, gdy bufor transmitujący i bufor odbierający są puste. Bit SPIF powinien być wyczyszczony przez odczytanie tego rejestru przerwań po zgłoszeniu przerwania.

**Inicjalizacja konfiguracji SSP:**

SSP\_CFG\_Type SSP\_ConfigStruct;

SSP\_ConfigStructInit(&SSP\_ConfigStruct);

SSP\_Init(LPC\_SSP1, &SSP\_ConfigStruct);

Struktura SSP\_CFG\_Type zawiera rejestry konfiguracyjne interfejsu SSP, takie jak CR0 i CR1. Funkcja SSP\_ConfigStructInit inicjalizuje tę strukturę do wartości domyślnych, ustawiając wszystkie rejestry konfiguracyjne na wartości zerowe. Następnie funkcja SSP\_Init inicjalizuje SSP1 (w tym przykładzie) poprzez zapisanie zawartości struktury konfiguracyjnej do odpowiednich rejestrów interfejsu SSP.

**Rejestry w SSP:**

1. Rejestr CR0 (Control Register 0):

* Bity 0-3, DSS (Data Size Select): Określa liczbę bitów danych przesyłanych w jednym transferze. Możliwe wartości to 4-16 bitów.
* Bity 4-5, FRF (Frame Format): Określa format ramki danych. Może to być SPI lub TI SSI lub Microwire.
* Bit 6, CPOL (Clock Polarity): Kontroluje stan aktywny zegara SCK. 0 oznacza, że SCK jest aktywne w stanie wysokim, a 1 oznacza, że SCK jest aktywne w stanie niskim.
* Bit 7, CPHA (Clock Phase): Kontroluje fazę zegara SCK w czasie transferu danych. 0 oznacza próbkowanie danych na pierwszym zboczu zegara, a 1 oznacza próbkowanie danych na drugim zboczu zegara.
* Bity 8-15, SCR (Serial Clock Rate): Określa preskaler dla generacji częstotliwości zegara SCK.

2. Rejestr CR1 (Control Register 1):

* Bit 0, LBM (Loop Back Mode) :Włącza tryb pętli zwrotnej, w którym dane są przesyłane z wyjścia do wejścia wewnętrznie w module SSP.
* Bit 1, SSE (Synchronous Serial Port Enable) :Włącza interfejs SSP.
* Bit 2, MS (Master/Slave Mode): Wybiera tryb pracy interfejsu SSP jako master lub slave.
* Bit 3, SOD (Slave Output Disable) : Wyłącza wyjście danych ze slave SSP, jeśli urządzenie pracuje w trybie slave.

3.CPSR (Clock Prescale Register): Określa preskalery dla generacji częstotliwości zegara SSP.

4.IMSC (Interrupt Mask Set/Clear Register): Umożliwia włączanie/wyłączanie maski przerwania dla konkretnych zdarzeń SSP.

5.RIS (Raw Interrupt Status Register): Przechowuje stan nieprzetworzonych przerwań.

6.MIS (Masked Interrupt Status Register): Przechowuje stan przerwań po uwzględnieniu maski.

7.ICR (Interrupt Clear Register): Służy do czyszczenia przerwań.

8.SR (Status Register): Zawiera informacje o stanie interfejsu SSP, takie jak flaga przepelnienia bufora, flaga zakończenia transmisji, flaga dostępności danych itp.

9.DR (Data Register): Służy do odczytu/zapisu danych.

10. DMACR (DMA Control Register): Kontroluje używanie DMA w SSP.

**Włączenie SSP:**

SSP\_Cmd(LPC\_SSP1, ENABLE);

Rejestr CR1 (Control Register 1) w interfejsie SSP zawiera bit kontrolny, który włącza lub wyłącza interfejs SSP. W tym przypadku, wywołanie funkcji SSP\_Cmd włącza SSP1 poprzez ustawienie bitu ENABLE w rejestrze CR1.

**4. Analiza skutków awarii**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ryzyko | Prawdopodo-  bieństwo | Znaczenie | Sposób wykrywania awarii | Iloczyn | Reakcja |
| Mikrokontroler | 0.1 | krytyczne (10) | Brak |  | Brak |
| Diody pca95xx | 0.3 | niegroźne | Brak |  | Brak |
| Wyświetlacz  OLED | 0.1 | krytyczne | Można podjąć próbę odczytu danych z ekranu.  Niepowodzenie oznacza awarię ekranu. | 2.0 | Brak |
| RTC | 0.1 | Średnie(5) | Brak | 0.5 | Brak |
| DAC | 0.1 | Niegroźne (brak dźwięku w trakcie wywoływania alarmu) | Brak | 0.1 | Brak |